CSE514 PROGRAMMING ASSIGNMENT 1

Instructor Cynthia Ma

Tri Pham

Date: 10/13/2021

I/ Introduction

1. Description

This assignment aims to strengthen the knowledge of regression models, specifically univariate and multivariate linear regression. Regression is a supervised learning algorithm that models the relationships between a dependent variable and one or more independent variables. More formally, given N data points $\{(x_1,y_1),(x_2,y_2), \ldots, (x_i, y_i), \ldots, (x_{N-1}, y_{N-1}), (x_N, y_N)\}$ where x_i is the i^{th} data point's independent variables, and y_i is the dependent variable, our model's goal is to estimate y correctly using x. For this assignment, we will be using the Concrete Compressive Strength dataset provided by the University of California, Irvine (UCI) Machine Learning Repository. There are 1030 samples in this dataset, and they are split into a training set that contains the first 900 samples and a testing set that contains the remaining 130 samples. Once the model is trained, it is used to predict the response variable of the testing set, thus providing observation for how well the model estimates y given unseen x. Since this assignment focuses on univariate and multivariate linear regression, the main approach in training a given dataset is to first define a cost function (or objective function). Then, we will use the gradient descent algorithm for optimization, namely minimizing our cost function as we update the model's parameters. Once the parameters are updated after training, we can test the trained model with the test dataset.

2. Formulation

Let us first define our univariate linear regression function:

$$f(x) = mx + b$$

where m is the coefficient of a single independent variable x and b is the bias term.

Next, we define our cost function to be Mean Square Error, formally written as:

$$C = \frac{1}{N}\sum_{i=1}^{N}(\mathrm{y_i} - (mx_i + b))^2$$

where N is the total number of samples being considered, y_i represents the true value of sample i, and $f(x_i) = mx_i + b$ denotes the estimated value of sample i.

Finally, in our gradient descent algorithm, we first find the gradient of the cost function with respect to parameter m and, similarly, the gradient of the cost function with respect to parameter b:

$$\frac{\partial C}{\partial m} = \frac{1}{N}\sum_{i=1}^{N} -2x_i\big(y_i - (mx_i + b)\big) \qquad \frac{\partial C}{\partial b} = \frac{1}{N}\sum_{i=1}^{N} -2\big(y_i - (mx_i + b)\big)$$

These parameters are updated iteratively with the help of a hyperparameter called the learning rate α. This hyperparameter influences how big of an update for both m and b at each iteration as the goal is to continue minimizing the cost function.

$$m_{new} = m_{old} - \alpha * \frac{\partial C}{\partial m} \qquad b_{new} = b_{old} - \alpha * \frac{\partial C}{\partial b}$$

Our stopping conditions are to either end our gradient descent algorithm after a maximum number of iterations or when the change in the cost function between two consecutive iterations is negligible (cost function convergence check).

For multivariate linear regression, the approach is essentially the same, except this time we are working with higher dimensions for our data. In particular, the input x is a vector of features with length p, and our model will now have p + 1 parameters (one bias term parameter b and eight coefficients where each corresponding to a feature). Regardless, we can formulate our equations slightly different using knowledge from matrix calculation in linear algebra:

$$y = f(x) = a\,.\,x$$

where a = $(b, a_1, \ldots, a_p)^T$ and x = $(1, x_1, \ldots, x_p)^T$. Essentially, we can estimate y by calculating the dot product between a and x. The parameter updating rules remain the same as the univariate linear regression where we still apply the gradient descent algorithm. Briefly, in each iteration we calculate the derivative with respect to each parameter in a, calculate the step size for each gradient to descent by, calculate the new parameters, check the cost function convergence with those parameters, and finally update the parameters if the change in the cost function between two consecutive iterations is not negligible. For multivariate linear regression, we want to be careful when working with matrices.

3. Implementation

This section includes my pseudocode for both univariate and multivariate linear regression. They follow the formulation presented above. There is one main note that I would like to highlight in my implementation. For the univariate regression model, I choose to store each parameter m and b in a list to observe my initial implementation. As I become more familiar with the model, I decide to implement the multivariate regression model without storing all the parameters while still ensuring the model's correctness.

a) Pseudocode 1: Univariate linear regression model

Input: An independent variable x, a dependent variable y, a learning rate α, a maximum iteration threshold T, an empty list L_m, an empty list L_b, and an empty cost list L_C.

Output: Non-empty list L_m, L_b, L_C and that contains updated parameters m, b, and cost C after each iteration for all iterations.

// We assume that x is normalized and there is a function helper *gradient* that calculates the derivative with respect to (wrt) m and the derivative wrt b for each iteration.

L_m.append(1); L_b.append(0); L_C.append(0); // append initial value of parameter m, b, and C

for $i = 0$ to $T - 1$ **do**

∇m, ∇b = *gradient*(i); // calculate the derivative wrt m and that wrt b for iteration i

$\delta m = \alpha * \nabla m$; // calculate step size for updating m and b

$\delta b = \alpha * \nabla b$;

$m_{new} = L_m[i] - \delta m$; // calculate new m, b, and C

$b_{new} = L_b[i] - \delta b$;

$C_{new} = \frac{1}{N}\sum_{j=1}^{N}\left(y_j - \left(m_{new} * x_j + b_{new}\right)\right)$;

if $| C[i] - C_{new} | < 10^{-5}$ **then** // check if the change in the cost function is negligeable

break

else // append the updated changes to L_m, L_b, and L_C

L_m.append(m_{new}); L_b.append(b_{new}); L_C.append(C_{new});

end

end

b) Pseudocode 2: Multivariate linear regression model

Input: A vector X of length p containing independent variables, a dependent variable y, a learning rate α, a maximum iteration threshold T, an empty list L_a and an empty cost list L_C.

Output: Non-empty list L_a that contains final updated parameter $a_1, \ldots, a_p$, and b after all iterations and cost list L_C containing updated cost C after each iteration for all iterations.

// We assume that X is normalized and 1 is concatenated to the start of X for association with b in L_a when doing matrix multiplication. There is a function helper *gradient* that calculates the derivative wrt to each parameter in L_a for each iteration.

L_a = [0] * (p + 1); // initialize L_a of size p + 1 where each index assigned to 0

L_C.append(0); // append initial value of C

for i = 0 to $T - 1$ **do**

    $\nabla L_a = gradient(i)$; // calculate the derivative wrt each a in L_a for iteration i

    $\delta L_a = \alpha * \nabla L_a$; // calculate step size for updating each a in L_a

    $L'_a = L_a - \delta L_a$; // update each a in L_a and calculate new C

    $C_{new} = \frac{1}{N}\sum_{j=1}^{N}(y - L'_a . X)$;

    if $| C[i] - C_{new} | < 10^{-5}$ **then** // check if the change in the cost function is negligible

        break

    else

        L_C.append(C_{new}); $L_a = L'_a$; // append the updated changes to L_C and update L_a

    end

end

II/ Results

As mentioned before, our cost function is Mean Square Error (MSE). This function also serves as our evaluation metric for this assignment. Thus, in this section, there will be nine MSE results for the training dataset (eight for univariate regression and one for multivariate regression) and another nine for the testing dataset. Additionally, there will be eight plots of the trained univariate models on top of scatterplots of their respective training data. **All models shown in this report have a learning rate of 0.1 and a max iteration threshold of 100**.

```
############ MSE results for all models on training data ################
Model 1: 231.5914898445244
Model 2: 290.78602993660553
Model 3: 295.6682192089344
Model 4: 292.142239689266
Model 5: 252.19555326566433
Model 6: 295.04221594318284
Model 7: 295.8157312596233
Model 8: 268.61033530547456
Model 9: 149.3618859868741
############ ------------------------------------------- ################
```

Fig1. Nine MSE results for the training dataset (model 1-8 each contains an independent variable and model 9 contains all independent variables from the concrete dataset)

```
############ MSE results for all models on testing data ################
Model 1: 79.00891650129259
Model 2: 169.1952768421031
Model 3: 160.1239124083324
Model 4: 158.9213932823104
Model 5: 237.86953543726565
Model 6: 160.33969298386683
Model 7: 160.9839926734348
Model 8: 148.18185515703445
Model 9: 98.5188363850372
############ ----------------------------------------- ################
```

Fig2. Nine MSE results for the test dataset

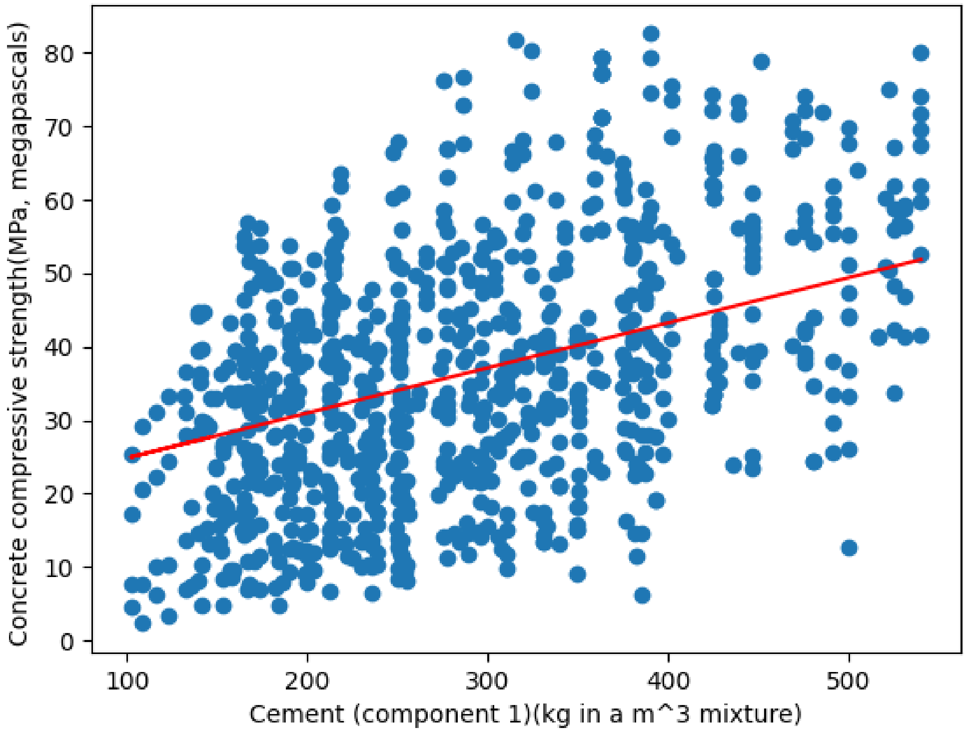

Fig3. Trained univariate **model 1** on top of scatterplots of its respective training data

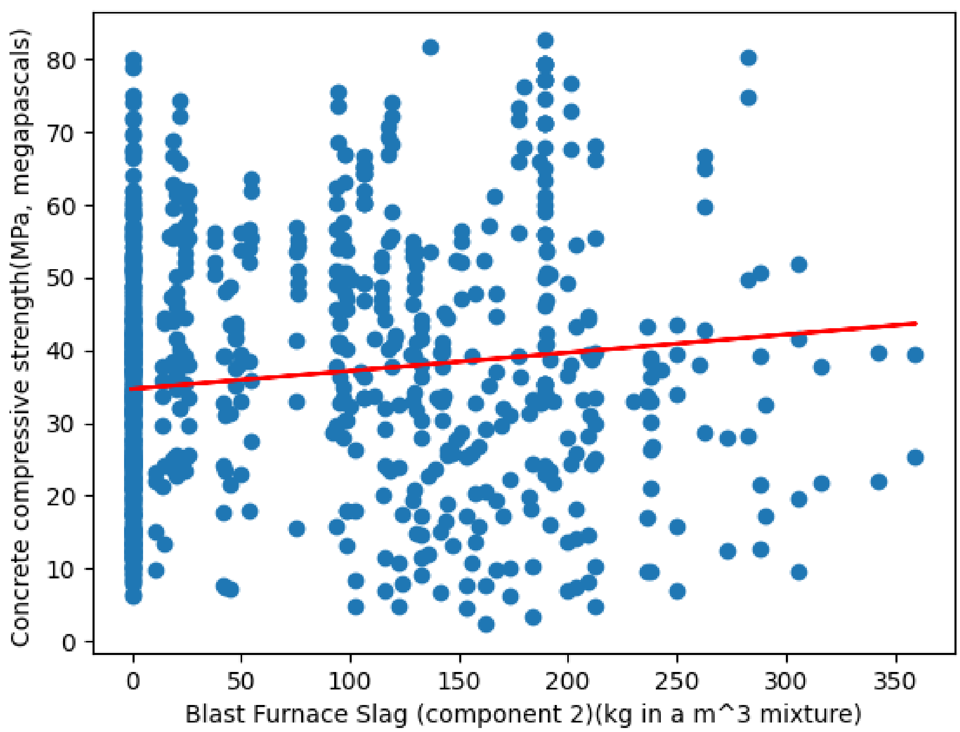

Fig4. Trained univariate **model 2** on top of scatterplots of its respective training data

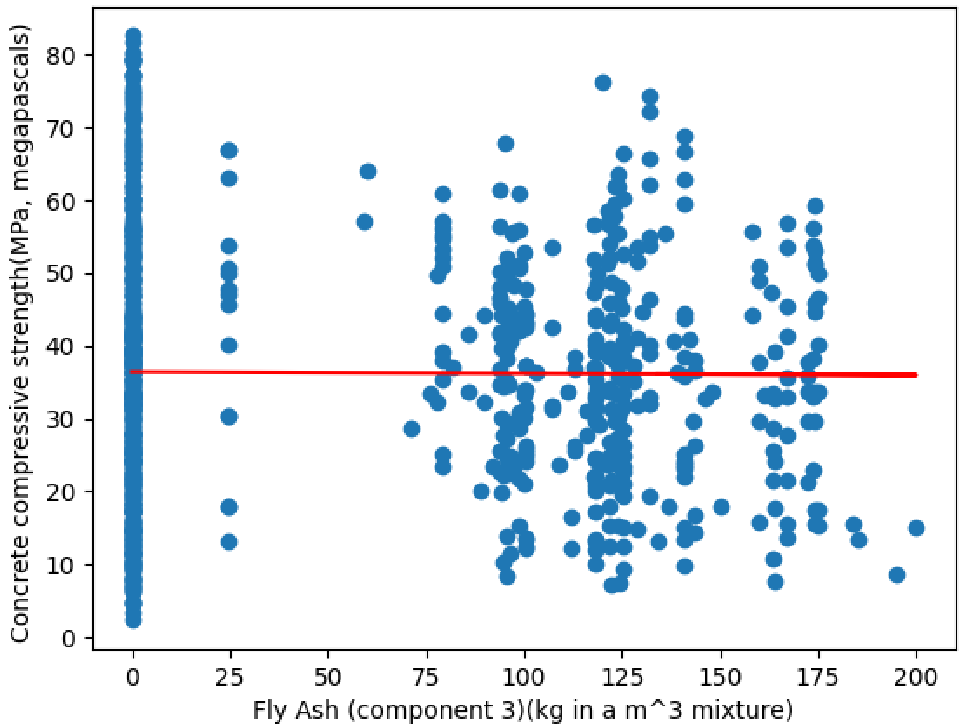

Fig5. Trained univariate **model 3** on top of scatterplots of its respective training data

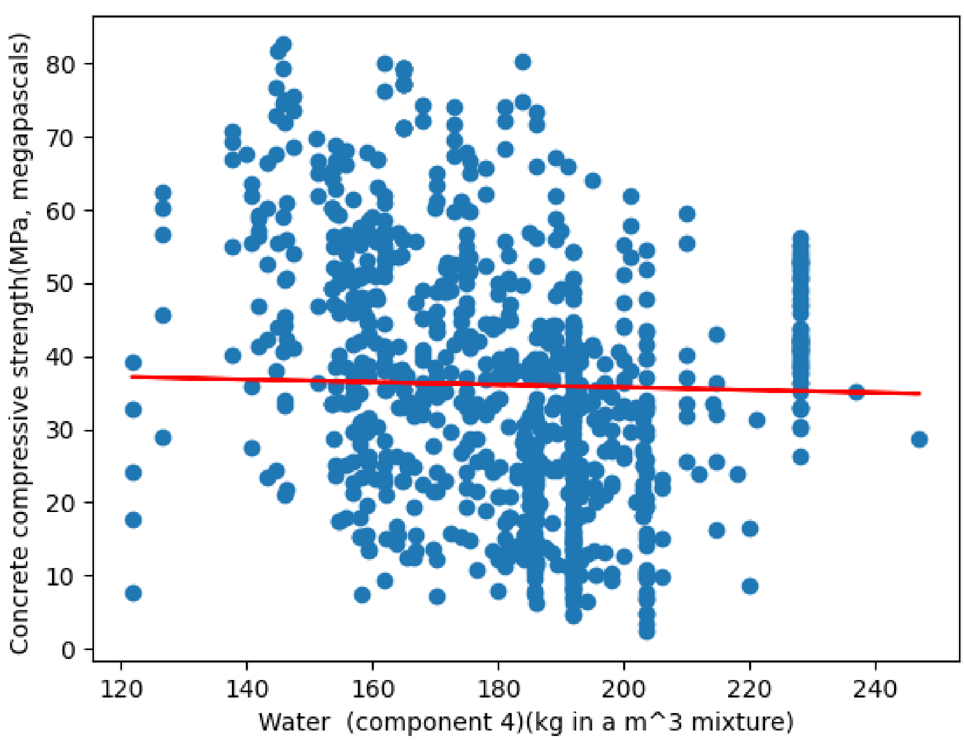

Fig6. Trained univariate **model 4** on top of scatterplots of its respective training data

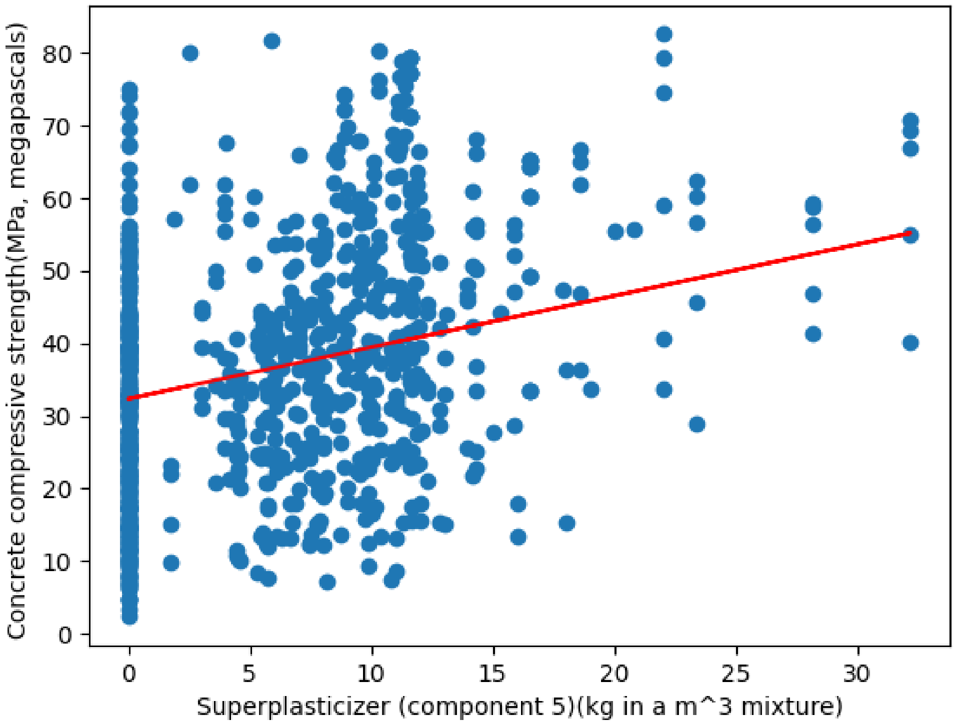

Fig7. Trained univariate **model 5** on top of scatterplots of its respective training data

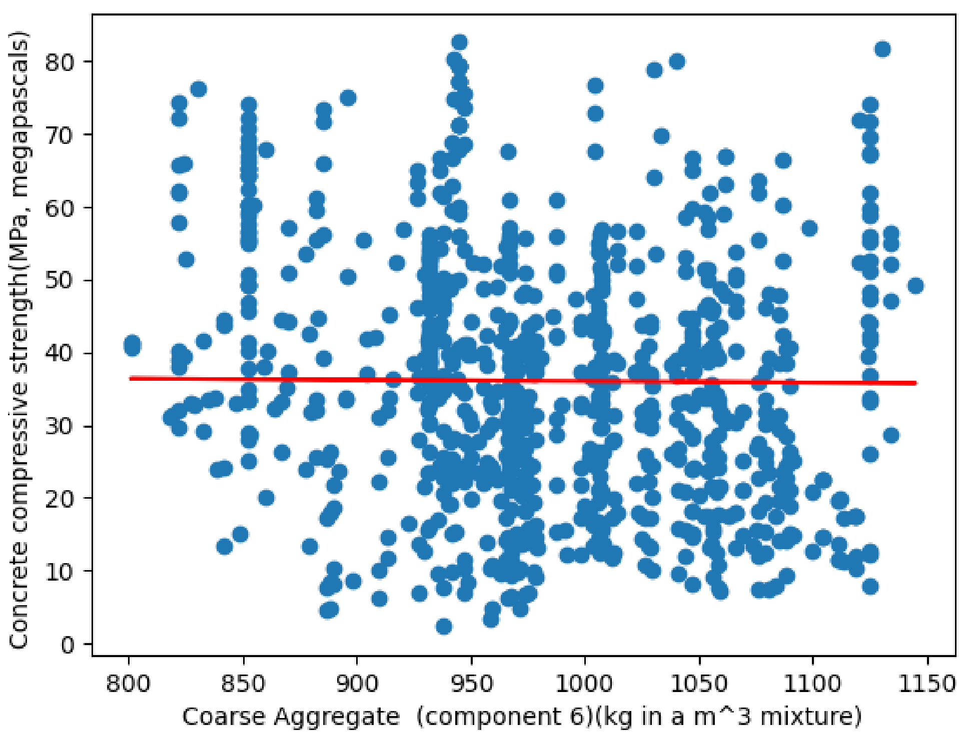

Fig8. Trained univariate **model 6** on top of scatterplots of its respective training data

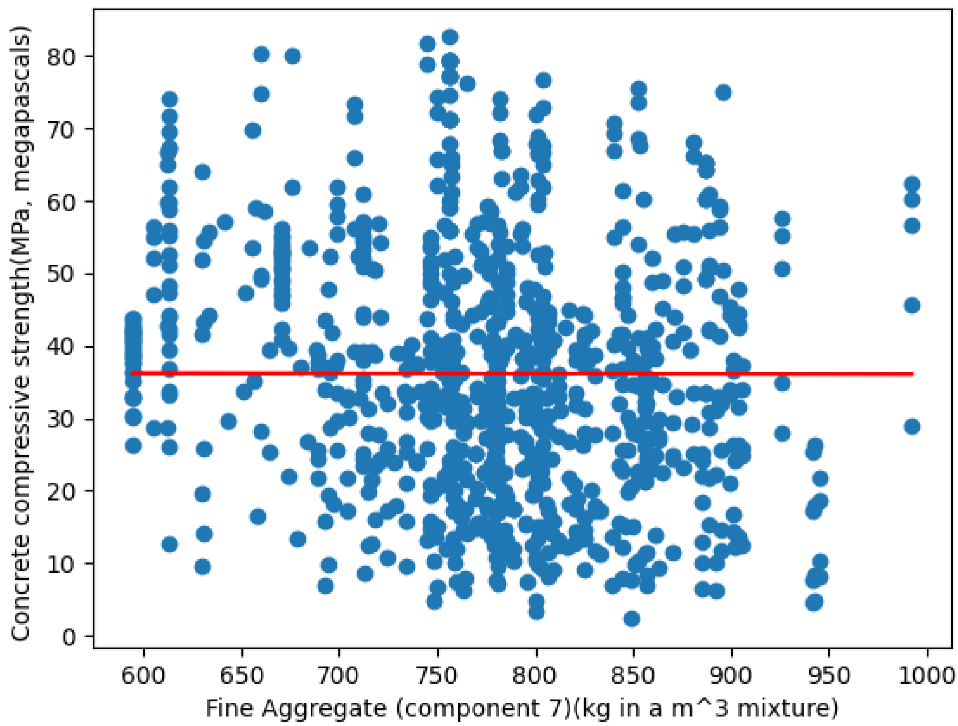

Fig9. Trained univariate **model 7** on top of scatterplots of its respective training data

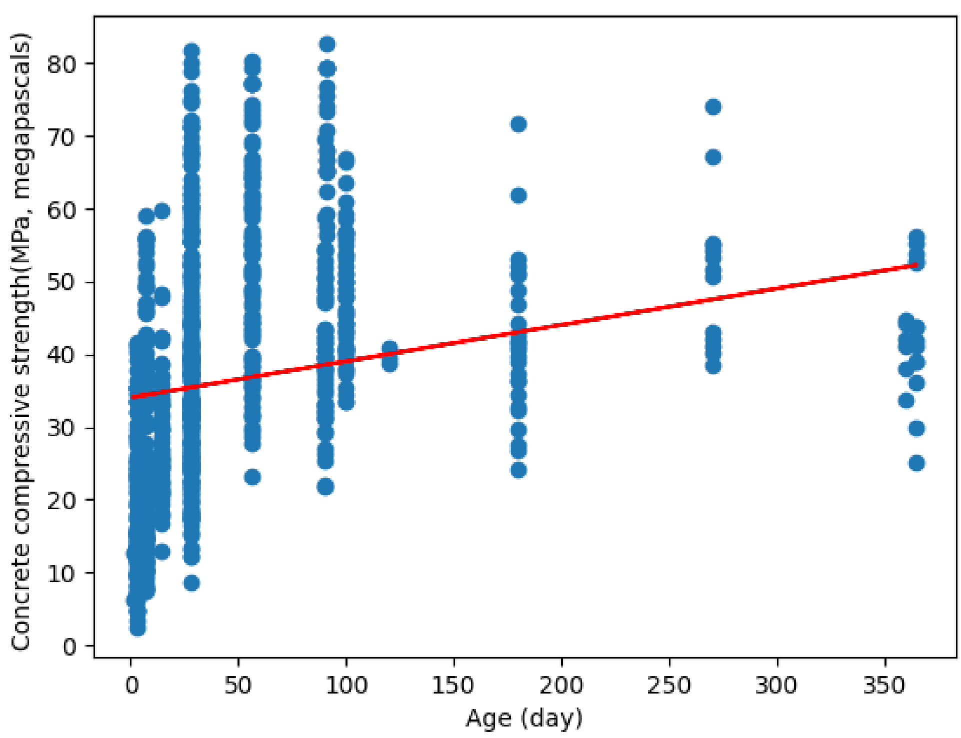

Fig10. Trained univariate **model 8** on top of scatterplots of its respective training data

III/ Discussion

Looking at the MSE results for the training dataset in Fig1., we see that with a **chosen learning rate of 0.1 and max iteration threshold of 100**, model 1 achieves the least MSE (≈231.59) out of all univariate models, followed by model 5 (≈252.19), model 8 (≈268.61), and model 2 (≈290.78). The remaining univariate models have roughly similar MSE results (≈295). Looking from the plots for all trained univariate models, since the independent variable in each model 1, 2, 5, and 8 has a positive correlation with the dependent variable, we can see that their respective lower MSE result also reflects the correlation. The multivariate model can train its parameters well and beats all univariate models by achieving an MSE of roughly 149.36. Since the multivariate model packs more information, it essentially takes into account more variables where a combination of those variables helps reduce the MSE further as the model trains. In general, we see that all MSE results fall into an acceptable range below 300.

Looking at the MSE results for the testing dataset in Fig2., we see that, surprisingly, model 1 achieves the least MSE (≈79) out of all models, followed by the multivariate model 9 (≈98.51) and other univariate models. Generally, the same model that performed well on the training data also does well on the testing data for all models. They all achieve good MSE results. One reason why model 1 achieves better MSE than model 9 does could be that in the testing data, the data values in variable *Cement* outweigh other data values. Since model 1 solely deals with *Cement*, this model predicts well. Notice that there are also factors such that the test data size, the learning rate, the max iteration threshold, etc. that could effect a model’s performance. Particularly, when rerunning the program with **a learning rate of 0.1 and max iteration threshold of 1000**, model 1 achieves an MSE around 82.25 while model 9 gets an MSE around 83.74.

Regarding whether the performance of the univariate models predicts or fails to predict which features are “more important” in the multivariate model, we can check if the parameters learned for a feature in its univariate model correlate with the parameters learned for a feature in the multivariate model.

```
############ Parameters learned for all models ################
Model 1: m = 26.884340128432406 , b = 24.963389508624328
Model 2: m = 8.981994248409771 , b = 34.66806041126327
Model 3: m = -0.45601893427297124 , b = 36.436872437429614
Model 4: m = -2.2567985339307204 , b = 37.11735975116177
Model 5: m = 22.82209450867729 , b = 32.354775723912965
Model 6: m = -0.6296384955754661 , b = 36.40383941461933
Model 7: m = -0.08634622347353575 , b = 36.170129914363145
Model 8: m = 18.17690088268158 , b = 34.07511346852229
Model 9: a = [17.00773562 27.68775533 12.99210844  4.13353053 -3.78282511 19.08301751
  0.45652607 -0.4377128  17.73828744] (first value represents bias b)
############ -------------------------------- ################
```

Fig11. Parameters learned for all models after training

From the figure, we notice that there are correlations between the parameters learned for a feature in its univariate model and those corresponding in the multivariate model. If we look at the trained univariate models' plots again, we remember that the independent variable in each model 1, 2, 5, and 8 has a positive correlation with the dependent variable. This can tell us that the features *Cement*, *Blast Furnace Slag*, *Superplasticizer*, and *Age* hold more importance than those remaining ones. Model 9 can also confirm this as the learned coefficients for the four features are relatively high (27.69, 12.99, 19.08, and 17.74). We know that the parameters' values do not completely match because the multivariate model is more complicated (requires a different learning rate, max iteration threshold, etc. to handle all variables), but the correlation results are essentially similar.

As mentioned above, *Cement*, *Blast Furnace Slag*, *Superplasticizer*, and *Age* are four main factors that represent concrete compressive strength. Out of the four, *Cement* shows the highest correlation with the concrete compressive strength. This also matches our normal intuition. When we talk about concrete, a lot of the time we will be thinking about cement because it is an essential component in creating concrete. In summary, we implement the gradient descent algorithm for univariate and multivariate linear regression models. Then, we train and test these models using the Concrete Compressive Strength dataset in the UCI repository. Finally, we observe the results, discuss the models' performance, and draw a conclusion to the factors that help predict concrete compressive strength. The code implementation can be found in the same directory as this report. Future work directions for this implementation are to include multivariate polynomial regression and sparse multivariate regression.